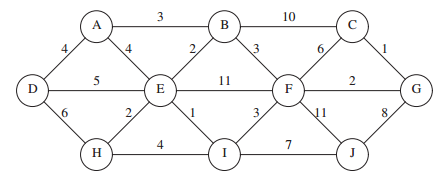
CS146-Lab10 Name: Michael Huang Due date: May 4th, 2016

Ex-1: Show that if unions are performed by height, then the depth of any tree is O(log(N)).

The height tree is almost the same as the size tree. Since the new unionized tree is placed into a tree that is at least twice as deep as before (mentioned in page 337), and the depth grows at log(N) pace. Since the find function takes O(log(N)), it can be said that the entire tree is at most log(N) depth.

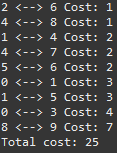
Ex-2: Write the code to implement the Kruskal’s minimum spanning tree. Test the code for the graph of figure 9.84 (page 420). The code also should display the total cost of all paths of this minimum spanning tree.



Code:

|  |
| --- |
| import java.util.\*;  public class Kruskal {  public static ArrayList<Edge> kruskal(List<Edge> edges, int numberOfVertices) {  DisjointSet set = new DisjointSet(numberOfVertices);  PriorityQueue<Edge> pq = new PriorityQueue<Edge>(edges);  ArrayList<Edge> minSpanTree = new ArrayList<Edge>();    while(minSpanTree.size() != numberOfVertices - 1) {  Edge edge = pq.remove();  int uset = set.find(edge.getVert1());  int vset = set.find(edge.getVert2());  if (uset != vset) {  minSpanTree.add(edge);  set.union(uset, vset);  }  }  return minSpanTree;  }    public static int totalCost(ArrayList<Edge> edges) {  int cost = 0;  for (Edge edge: edges) {  cost += edge.getWeight();  }  return cost;  }    public static void main(String[] args) {  ArrayList<Edge> edges = new ArrayList<Edge>();  final int A = 0;  final int B = 1;  final int C = 2;  final int D = 3;  final int E = 4;  final int F = 5;  final int G = 6;  final int H = 7;  final int I = 8;  final int J = 9;  edges.add(new Edge(A, B, 3));  edges.add(new Edge(A, D, 4));  edges.add(new Edge(A, E, 4));  edges.add(new Edge(B, C, 10));  edges.add(new Edge(B, E, 2));  edges.add(new Edge(B, F, 3));  edges.add(new Edge(C, F, 6));  edges.add(new Edge(C, G, 1));  edges.add(new Edge(D, E, 5));  edges.add(new Edge(D, H, 6));  edges.add(new Edge(E, F, 11));  edges.add(new Edge(E, H, 2));  edges.add(new Edge(E, I, 1));  edges.add(new Edge(F, G, 2));  edges.add(new Edge(F, I, 3));  edges.add(new Edge(F, J, 11));  edges.add(new Edge(G, J, 8));  edges.add(new Edge(H, I, 4));  edges.add(new Edge(I, J, 7));  ArrayList<Edge> mintree = kruskal(edges, 10);  for (Edge edge: mintree) {  System.out.println(edge.toString());  }  System.out.println(totalCost(mintree));  }  }  class Edge implements Comparable<Edge> {  private int u;  private int v;  private int weight;    public Edge(int vertex1, int vertex2, int weight) {  u = vertex1;  v = vertex2;  this.weight = weight;  }    public int getVert1() {  return u;  }    public int getVert2() {  return v;  }    public int getWeight() {  return weight;  }  public int compareTo(Edge edge) {  if (this.weight < edge.getWeight()) {  return -1;  } else if (this.weight > edge.getWeight()) {  return 1;  } else {  return 0;  }  }    public String toString() {  return u + " --> " + v + " Cost: " + weight;  }  }  class DisjointSet {  private int[] set;  public DisjointSet(int numberOfElements) {  this.set = new int[numberOfElements];  for (int i = 0; i < set.length; i++) {  set[i] = -1;  }  }  public void union(int root1, int root2) {  if (set[root2] < set[root1]) {  set[root1] = root2;  } else {  if (set[root1] == set[root2]) {  set[root1]--;  }  set[root2] = root1;  }  }  public int find(int x) {  if (set[x] < 0) {  return x;  } else {  return set[x] = find(set[x]);  }  }  } |

Console:



Ex-3: Section 8.7 described the generating of mazes. Suppose we want to output the path in the maze. Assume that the maze is represented as a matrix; each cell in the matrix stores information about what walls are present (or absent).

a. Write a program that computes enough information to output a path in the maze. Give output in the form SEN... (representing south, then east, then north, etc.).

Code:

|  |
| --- |
| import java.util.\*;  public class Maze {  class Cell {  boolean north, east, south, west = false;  int number;  public Cell(int number) {  this.number = number;  }  }  private Cell[][] cells;  private DisjointSet set;  int prev;  public Maze(int rows, int cols) {  cells = new Cell[rows][cols];  set = new DisjointSet(rows \* cols);  int count = 0;  int prev = 0;  for (int i = 0; i < cells.length; i++) {  for (int j = 0; j < cells[0].length; j++) {  cells[i][j] = new Cell(count);  }  }  for (int i = 0; i < cells.length; i++) {  for (int j = 0; j < cells[0].length; j++) {  set.union(maze[i][j].number, maze[i++][j]);  }  }  }    public String getPath() {  return "";  }    public static void main(String[] args) {  Maze maze = new Maze(2, 2);  System.out.println(maze.getPath());  }  } |

(Extra Credit)

b. Write a program that draws the maze and, at the press of a button, draws the path.